Chapter 10 Dependable systems:

10.1 Dependability properties

10.2 Socio-technical systems

10.3 Redundancy and diversity

10.4 Dependable processes

10.5 Formal methods and dependability

Note: Images excluded due to time constraints

10.1 Dependability properties

Overview

- Dependability in computer systems refers to the trustworthiness and reliability with which a system operates.
- It is a complex property that cannot be quantified numerically but is usually described in relative terms like "not dependable," "very dependable," and "ultra-dependable."

Principal Dimensions of Dependability

1. **Availability**: The likelihood that a system will be operational and capable of providing useful services at any given time.
2. **Reliability**: The probability that a system will perform as expected over a given period.
3. **Safety**: A measure of the system's likelihood to cause harm to people or its environment.
4. **Security**: The system's ability to resist accidental or deliberate intrusions.
5. **Resilience**: The system's capability to maintain service continuity during disruptive events like equipment failure and cyberattacks.

Sub-properties of Dependability

- Security encompasses "integrity" and "confidentiality."
- Reliability includes "correctness," "precision," and "timeliness."

Related Properties

1. **Repairability**: The speed with which a system can be repaired after a failure.
2. **Maintainability**: The ease with which the system can adapt to new requirements without introducing new errors.
3. **Error Tolerance**: The system's ability to avoid and handle user input errors.

Factors for Building Dependable Systems

1. Avoid accidental errors during system specification and development.
2. Implement effective verification and validation processes.
3. Design systems to be fault-tolerant.
4. Implement protection mechanisms against external attacks.
5. Properly configure the system for its operating environment.
6. Include capabilities to recognize and resist external cyberattacks.
7. Design for quick recovery from system failures and cyberattacks without data loss.

Performance Trade-offs

- Designers often face trade-offs between system performance and dependability, particularly when adding redundant code for fault tolerance.

Costs and Challenges

- Increasing a system's dependability often results in higher costs for design, implementation, and validation.
- As systems become more dependable, the cost of further improvement increases, especially for ultra-dependable systems like safety-critical control systems.
- Extensive testing is required to demonstrate dependability, which can further escalate costs.

10.2 Socio-technical systems:

Overview

- Sociotechnical systems are complex structures that include not just hardware and software, but also people, processes, organizations, and societal elements.
- Software engineering should be viewed as an integral part of systems engineering, as software is deeply interconnected with other layers of the system.

Characteristics and Layers

- A system is more than the sum of its parts, having properties that only appear when all components work together.

- Sociotechnical systems can be viewed as layers in a stack:
    i. **Equipment Layer**: Consists of hardware devices, including computers.
    ii. **Operating System Layer**: Interacts with hardware to provide common facilities.
    iii. **Communications and Data Management Layer**: Extends the OS and allows for broader functionalities like access to remote systems and databases (also known as middleware).
    iv. **Application Layer**: Provides application-specific functionalities.
    v. **Business Process Layer**: Includes organizational processes that utilize the software system.
    vi. **Organizational Layer**: Contains higher-level strategies, rules, and norms.
    vii. **Social Layer**: Involves societal laws and regulations governing the system.

Interactions and Complexities

- Ideally, interactions should be between neighboring layers, but real-world systems often have unexpected cross-layer interactions.
- Changes in one layer (e.g., law changes in the social layer) can trigger modifications in other layers.

System-Level Considerations

- A holistic view is essential for software security and dependability.
- Software failures can have cascading effects on the physical and human environment.
- Strategies must be in place to contain software failures within layers and to understand how failures in one layer can impact others.

Software as a Solution and Problem

- Software is often the go-to for solving unexpected problems in the system but can also become the point of failure when stretched beyond its capabilities.
- Many software failures are not due to software itself but result from trying to adapt the software to changing system requirements.

Challenges and Examples

- Software engineers often face challenges in enhancing software capabilities without increasing costs.
- Examples include failures like the Denver airport baggage system, where software was expected to compensate for equipment limitations.

10.2.1 Regulation and Compliance

Overview

- The prevalent economic model globally involves privately owned companies offering goods and services in a competitive environment.
- Governments impose **regulations** to ensure the safety and security of citizens, limiting how these companies operate.

Role of Government

1. **Standard Setting**: Governments define rules and regulations that set the safety and security standards companies must meet.
2. **Regulatory Bodies**: Governments establish regulators with the power to enforce these standards.
    - Regulators can impose fines and even imprison company directors for breaches.
3. **Licensing**: In certain industries like aviation and nuclear energy, regulators issue licenses that a system must obtain before going operational.

Compliance Procedures

1. **Safety Case**: Companies must produce an extensive safety case to achieve certification.
    - This case should prove that the system can operate safely according to set regulations.
    - The process is costly and can be as expensive as developing the system itself.
2. **Sociotechnical System**: Compliance is not limited to just the software but applies to the sociotechnical system as a whole.
3. **Industry-Specific Concerns**: For example, in the nuclear industry, the safety case should address multiple elements like software protection systems, operational processes, and structural integrity.

Alternative Safety Mechanisms

- The safety case must also demonstrate that alternative safety mechanisms exist, which do not rely on software, in case the primary software protection system fails.

10.3 Redundancy and Diversity

Overview

- Component failures in systems are inevitable due to human errors, software bugs, and hardware degradation.
- Designing systems to be resilient against individual component failures is crucial, and this is often achieved through **redundancy** and **diversity**.

Key Strategies

1. **Redundancy**: Incorporates spare capacity in a system to take over if a part of the system fails.
    - Commonly used in everyday scenarios like multiple locks for home security.
2. **Diversity**: Ensures that redundant components are of different types to reduce the likelihood of similar failures.
    - Utilized in systems to avoid a single point of failure.

Practical Applications

1. **Data Backups**: Maintaining redundant copies of data on diverse external devices to mitigate disk failure.
2. **Redundant Components**: Inclusion of secondary components that provide the same functionality as primary components.
    - Switched in if the primary component fails.

3. **Checking Code**: Additional code that can detect issues like data corruption, which isn't strictly necessary for basic functionality but enhances reliability.

Complexity and Risks

- Incorporating redundancy and diversity adds complexity, making it harder to write and check code.
- Complexity can introduce new bugs and make existing ones harder to find.

Industry Examples

1. **Ariane 5 Rocket**: Failure attributed to lack of diversity in backup systems, leading to identical failure modes.
2. **Airbus 340 vs Boeing 777**: Airbus employs both diversity and redundancy, while Boeing focuses on extensive validation and simplicity.

Dependable Development Processes

- Activities like software validation should not rely on a single tool or technique to avoid process failure.
- Team members may provide diverse perspectives based on their unique skills and experiences.

Human Factors in Dependable Processes

- Systems should be designed to detect human errors and allow corrections to enhance overall dependability.

By incorporating redundancy and diversity, systems can better ensure dependability, although this comes with its own challenges and complexities.

10.4 Dependable Processes

Overview

- **Dependable software processes** aim to produce dependable software, containing fewer errors and therefore less prone to failure.
- Using a dependable process aids in convincing regulators of the quality of software engineering practice.

Characteristics of Dependable Processes

1. **Explicitly Defined**: The process is clearly outlined through a process model.
    - Data is collected to show adherence to the process model.
2. **Repeatable**: The process is consistent across projects and teams, minimizing individual interpretation.
3. **Auditable**: The process is understandable to external parties who can verify its standards.
4. **Diverse**: Includes diverse and redundant verification and validation activities.
5. **Documentable**: Has a defined model outlining activities and necessary documentation.
6. **Robust**: Can recover from failures in individual activities.
7. **Standardized**: Complies with a comprehensive set of software development standards.

Key Activities in Dependable Processes

1. **Requirements Reviews**: Ensure completeness and consistency.
2. **Requirements Management**: Manages changes and their impact.
3. **Formal Specification**: Utilizes mathematical models for detailed requirements analysis.
4. **System Modeling**: Explicitly documents the software design.
5. **Design and Program Inspections**: Multiple people inspect system descriptions.
6. **Static Analysis**: Automated checks on source code for anomalies.
7. **Test Planning and Management**: Designs and manages comprehensive system tests.

Quality and Change Management

- Effective quality management and change management processes are universally necessary.
- **Configuration Management**: Ensures the correct components are included in system builds.

Agile Approaches in Dependable Processes

- Although traditional dependable processes are plan-based, there's growing interest in incorporating **agile techniques**.
- Agile techniques can be formally defined and documented to meet the needs of dependable systems.

By focusing on these characteristics and activities, dependable processes aim to produce robust, error-free software. This is particularly crucial for systems that require regulatory approval and those that have long development cycles.

10.5 Formal Methods and Dependability

Overview

- **Formal methods** are mathematical approaches to software development aimed at increasing dependability.
- They have been successfully applied in various domains like train control, cash card, and flight control systems.

Core Concepts

1. **Formal Model**: Establishes a mathematical representation of the software.
    - Enables formal analysis to search for errors and inconsistencies.
2. **Proofs and Theorem Proving**: Initially manual, now supported by automated software, but still a specialized task.
3. **Refinement-Based Development**: Uses trusted transformations to generate a program that's consistent with its formal specification.
4. **Model-Checking**: Checks properties of a formal state model of the system.
    - Either confirms a system property or provides a counterexample.

Types of Errors Addressed

1. **Specification and Design Errors**: Formal methods help in revealing errors and omissions in software requirements.
2. **Inconsistencies**: Formal methods can find discrepancies between a specification and a program.

Advantages of Formal Methods

1. **Deep Understanding of Requirements**: Formal specification forces a detailed analysis of the requirements.
2. **Automated Analysis**: Formal languages allow for automated inconsistency and incompleteness checking.
3. **Correctness**: Methods like the B method transform the formal specification into a program, ensuring it meets its specification.
4. **Reduced Testing Costs**: Verification against the specification can minimize the need for extensive testing.

Limitations and Challenges

1. **Complexity**: Difficult for non-experts to understand, posing a challenge in verifying that it represents domain requirements accurately.
2. **Cost Estimation**: Uncertainty in quantifying the cost savings from using formal methods.
3. **Lack of Training**: Most software engineers are not trained to use formal specification languages.
4. **Scalability**: Difficult to apply to very large systems.
5. **Tool Support**: Limited and often challenging to use.
6. **Incompatibility with Agile**: Not easily compatible with incremental agile development approaches.

Despite their promise, formal methods have seen limited adoption in the industry due to various challenges, including a lack of clear demonstration of cost-effectiveness compared to other software engineering methods.

Summary

- System dependability is important because failure of critical computer systems can lead to large economic losses, serious information loss, physical damage or threats to human life.
- The dependability of a computer system is a system property that reflects the user’s degree of trust in the system. The most important dimensions of dependability are availability, reliability, safety, security, and resilience.
- Sociotechnical systems include computer hardware, software, and people, and are situated within an organization. They are designed to support organizational or business goals and objectives.
- The use of a dependable, repeatable process is essential if faults in a system are to be minimized. The process should include verification and validation activities at all stages, from requirements definition through to system implementation.
- The use of redundancy and diversity in hardware, software processes, and software systems is essential to the development of dependable systems.
- Formal methods, where a formal model of a system is used as a basis for development, help reduce the number of specification and implementation errors in a system. However, formal methods have had a

limited take-up in industry because of concerns about the cost-effectiveness of this approach.